Automated Program Debugging Tool

Background of the Invention

a. Field of the Invention

The present invention pertains to computer program development tools and specifically to automated tools for facilitating debugging of computer programs.

b. Description of the Background

Testing and debugging computer code is typically a large portion of the development cycle of a computer program. Several program development environments are specifically constructed to aid a developer in the writing, editing, testing and debugging of code by enabling the developer to step into and through the code, evaluate variable values, etc. Such development environments are not always available to those developers whose applications are not supported by the environments.

It is commonplace for software developers to insert print statements at periodic intervals throughout their program so that the state of the program may be monitored when the program is run. These print statements are time consuming to write and may require a working knowledge of the function of the code to place the statements in the proper locations.

It would therefore be advantageous to provide a system and method for automatically placing debugging statements into a program that may enable a tester or developer to efficiently trace the flow of the program and help in debugging the program. It would be additionally advantageous if such a system and method were adaptable to different depths of debugging and different debugging techniques.

Summary of the Invention

The present invention overcomes the disadvantages and limitations of the prior art by providing a system and method for placing output statements at periodic intervals throughout the source code of a computer program. The computer program is analyzed for proper locations for output statements which may have different levels of verbosity that are controllable within the source code and by the debugging program.

In a low verbosity level, the source code may be analyzed to find each function call or subroutine, wherein an output statement may be inserted. In a high verbosity level, an output statement may be inserted for each functional line of the source code. In some embodiments, the logic levels of certain statements or variable values may be included in the output statements.

When the source code is executed with the embedded output statements, a log file may be generated that aids the developer to trace the actual execution of the program at various levels of detail. The log file may contain the necessary information to enable the developer to quickly locate the specific line of code in a specific file that was being executed at a particular point in time.

The present invention may therefore comprise a computerized method for adding debugging statements to a computer source code having a plurality of lines of code comprising: creating a first output file; setting a verbosity level to a predetermined level; traversing through the computer source code by reading and analyzing a portion of the source code at a time, the reading and analyzing comprising: reading the portion of the source code, the portion comprising executable statements and comments; and if the portion comprises an executable statement, writing the executable statement to the output file, constructing an output statement comprising at least an indicator of the location of the executable statement within the source code, and writing the output statement to the output file; and causing the output file to be executed in place of the computer source code.

The present invention may further comprise a computer program for adding debugging statements to a computer source code having a plurality of lines of code comprising: a first routine for creating a first output file; the ability to set a verbosity level to a predetermined level; a second routine for traversing through the computer source code by reading and analyzing a portion of the source code at a time, the reading and analyzing comprising: reading the portion of the source code, the portion comprising executable statements and comments; and if the portion comprises an executable statement, writing the executable statement to the output file, constructing an output statement comprising at least an indicator of the location of the executable statement

within the source code, and writing the output statement to the output file; and wherein the output file may be executed in place of the computer source code.

The advantages of the present invention are that a computer program source code may be automatically modified on a selective basis that greatly facilitates debugging and tracing of a computer program. The system and method may operate on source code that comprises a plurality of source code files. The modified computer source code may provide a trace file that identifies the specific location that was executed and may optionally provide various key values and variables when required, thus greatly speeding up code testing and debugging activities.

Brief Description of the Drawings

In the drawings,

FIGURE 1 is an illustration of an embodiment of the present invention showing a overall method for generating annotated source code.

FIGURE 2 is an illustration of an embodiment of the present invention showing a method for annotating source code.

FIGURE 3 is an illustration of a snippet of source code prior to having the annotated lines added thereto.

FIGURE 4 is an illustration of an annotated version of the snippet of code from FIGURE 3.

FIGURE 5 is an illustration of an output file that is generated when the source code of FIGURE 4 is executed.

Detailed Description of the Invention

Figure 1 illustrates an embodiment 100 of the present invention showing an overall method for generating annotated source code. The original source code 102 may comprise several files. The source code 102 may be compiled and executed 104 to create the normal output 106. In order to debug the source code 102, an automated code trace annotator 108 may generate annotated source code 110 that may be compiled and executed 112 using the same compiler as in block 104. The annotated source code may output the normal output 114 in addition to an execution log 116.

The embodiment 100 provides a way for source code 102 to be annotated such that code tracing and debugging can be efficiently performed. By automating the process of annotating the code, a developer may quickly determine how the program executes in order to determine if the program is operating as intended.

The automated code trace annotator 108 systematically steps through the source code 102 and adds output statements that create the execution log 116 when the annotated code is executed. The output statements may indicate function calls, specific lines of the code that are executed, original source code file names, variable values, or other information as may be deemed necessary.

The embodiment 100 allows the source code 102 to be executed with a series of annotation output statements while not adding the annotation output statements to the final version of the source code 102. In other words, a debugging scenario may operate as follows: the source code 102 may be annotated by the automated code trace annotator 108 and executed to determine a portion of the source code 102 to change. Those changes may be incorporated into the source code 102, the automated code trace annotator 108 may be re-executed, and the modified source code may be evaluated. Using this scenario, the source code 102 may be continually modified without having to add and remove debugging statements.

The method 100 may be adapted to virtually any type of computer programming environment and language. The method 100 is particularly suited to those programming environments where powerful debugging tools may not have been developed or where such tools are incapable of tracing and debugging various aspects of programs developed in such environments.

The automated code trace annotator 108 may have different levels of verbosity that may be set to have various types of information shown in the execution log 116 at runtime. For example, a low verbosity level may cause only high level function calls to be added to the execution log 116. A medium verbosity level may cause each executable statement to be written to the execution log 116. A high verbosity level may cause each variable and value of each variable to be written to the execution log 116 in addition to each executable statement. Various verbosity levels and attributes may be created by

those skilled in the arts while maintaining within the spirit and scope of the present invention.

The automated code trace annotator 108 may be a standalone program or may be incorporated into a compiler, interpreter, or other program that is used to execute the source code 102.

Figure 2 illustrates an embodiment 200 of the present invention showing a method for annotating source code. The method begins in block 202. The verbosity level is set in block 204. The source code file is opened for reading in block 206 and the modified source code file is opened for writing in block 208. A portion of the source code file is read in block 210. If an end of file is encountered in block 212, the modified source code file is closed in block 214 and if another file is to be annotated in block 216, the process resumes at block 206, otherwise the process ends in block 218. If the end of file was not encountered in block 212, the portion of code is evaluated to see if it is an executable statement in block 220.

If the portion of code is not an executable statement in block 220, then it is further analyzed in block 222 to see if an embedded command in a comment block changes the verbosity. If the verbosity is to be changed in block 224, the level is changed in block 226, otherwise, the process resumes at block 210.

If the portion of code is an executable statement in block 220, block 228 evaluates whether an output statement should be generated. If an output block is needed in block 228, it is generated in block 230, otherwise an executable statement is written in block 232 and the process resumes at block 210.

The portion of source code that is read in block 210 may be a single line or statement. In some programming languages, a single statement may span multiple lines. In other languages, multiple statements may be placed in a single line while still other languages require only one statement per line. The specific syntax of the language may have specific rules for statement and command construction that may need to be consulted when reading a useable portion of the source code file.

The embodiment 200 may operate on several files sequentially. For example, it may be common practice to place all the source code files for a specific programming project into a single directory. The embodiment 200 may be adapted to search through

each file of the directory and create a corresponding annotated version of that source code in that directory or a separate one.

In some embodiments, the annotated files may be given a new file name, while in other embodiments, the newly annotated files may be kept in a separate directory. Those skilled in the arts will appreciate that changing the filenames of included or linked files may present some challenges to the developer and thereby may be less desirable when an embodiment is directed towards applications with multiple linked files.

The comment areas of the source code may include embedded commands that serve to change the verbosity level at different points throughout the annotation process. For example, a frequently called and highly tested subroutine or function may have a low verbosity level that only includes an output line that indicates that the subroutine or function was called. In another area of the source code, embedded commands in the comment areas may be used to increase the verbosity level for code that is not thoroughly tested.

The output statement that is written in block 230 may include an indicator to the location of the program execution. The indicator may include the file name and line number of the executed statement or any other indicator as one skilled in the art may desire.

The output statement may further comprise a copy of the executed command for that portion of source code. The executed command may have variables and their values included in the output statement to further aid the developer in debugging the program.

Figure 3 illustrates a snippet of source code 300 prior to having the annotated lines added thereto. The source code example is in the C language.

Figure 4 illustrates an annotated version 400 of the snippet of code from Figure 3. The output statements include the "fprintf" statements located in many locations through the code.

Figure 5 illustrates an output file 500 that is generated when the source code of Figure 4 is executed. For each line of the output file 500, there is a source file name 502, a line number 504, and the executed statement 506.

The output file 500 provides a developer with a simple way to trace the function of the source code as well as provides the necessary reference information to locate a line

of code that the developer may wish to modify. In the case where the program abnormally terminates execution, the last executed line of code plus the file and line number of that source code will be the last line of the output file. By analyzing the output file directly, the developer may trace the logic of the code in order to identify areas of the source code to modify.

In some embodiments, the output file 500 may include variable listings or may display the various expressions that are evaluated in that line of code. For example, in the case of an IF statement, the output file 500 may include the value of the expression within the IF statement by some mechanism. Such a mechanism may include providing a "T" or "F" indicator at the beginning or end of the line. Other mechanisms include changing the color or other visual indicator of the text in the output file. Those skilled in the arts may use various methods of indication for expressions and variable values while keeping within the spirit and intent of the present invention.

The foregoing description of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed, and other modifications and variations may be possible in light of the above teachings. The embodiment was chosen and described in order to best explain the principles of the invention and its practical application to thereby enable others skilled in the art to best utilize the invention in various embodiments and various modifications as are suited to the particular use contemplated. It is intended that the appended claims be construed to include other alternative embodiments of the invention except insofar as limited by the prior art.